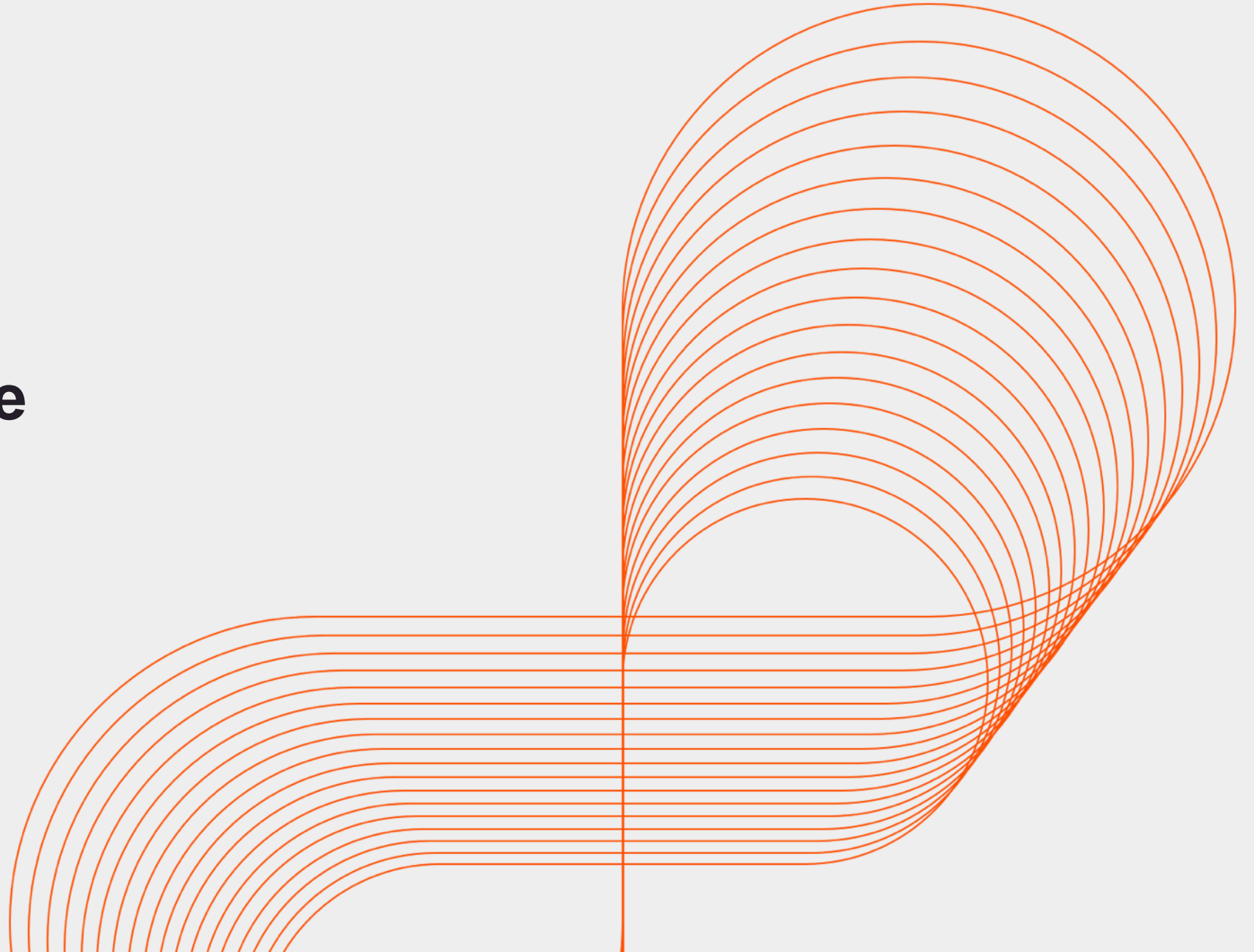




Persistent

Core Java: Object Lifetime

Persistent University



Objectives :

- **Object creation and initialization**
- **Memory Organization**
- **Garbage Collection**

Object creation and initialization

A decorative orange graphic consisting of a horizontal line that extends from the left edge of the slide, a vertical line that extends downwards from the horizontal line, and a large circle that is tangent to the horizontal line at its rightmost point and the vertical line at its bottom point.

Object

- An object is a chunk of memory. It is bundled with the code that manipulates memory.
- In memory, an object preserves its state and keeps on changing and evolving through out its life cycle.
- The life cycle of object involves three phases.
 - Declaration
 - Initialization
 - Destruction

Object declaration

- In this phase the variable of type class is declared.
- This is also termed as reference of a class.

```
public class Box{  
    private float height;  
    private float width;  
}  
  
public class BoxTest{  
    public static void main(String args[]){  
        Box chocolateBox; //reference of type Box  
    }  
}
```

Object instantiation

- The new operator instantiates an object by performing memory allocation for it.
- The new operator is followed by an argument that is a constructor of that class.

```
public class BoxTest{  
    public static void main(String args[]){  
        Box chocolateBox; //reference of type Box  
        chocolateBox = new Box();  
    }  
}
```



Constructor is invoked

Object initialization

- The constructors provided by classes, perform the operation of initialization of the object.
- These will initialize the instance data members for that object. These data members represent the state of that object.
- The Java virtual machine (JVM) allocates memory on the heap for the object i.e. instance data members of that object.
- Java makes sure that the memory is initialized to the default values of instance data members, before it is used by any code.

Memory Organization

- The two types of memory used in Java are :
 - Stack memory : stores primitive types and the addresses of objects.
 - Heap memory : stores object values.
- An object reference on the stack is an address and it refers to the memory location on heap where that object is stored.

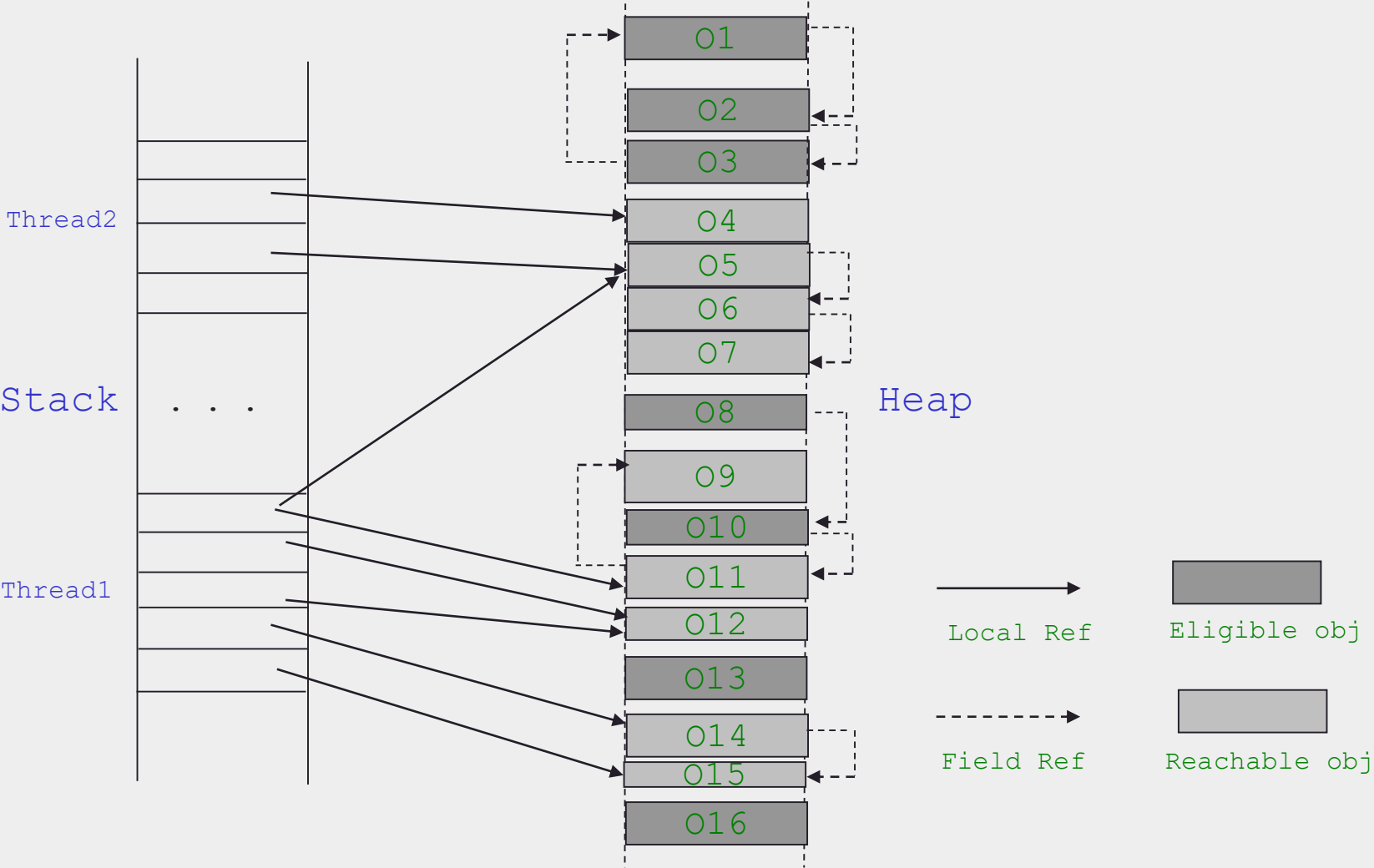
Reachable References

- Any reference that makes an object reachable.
- Reachable Object
 - if object is denoted by any local reference in a stack.
- Eligible Object
 - an unreachable object waiting for its memory to be reclaimed.

Memory Organization

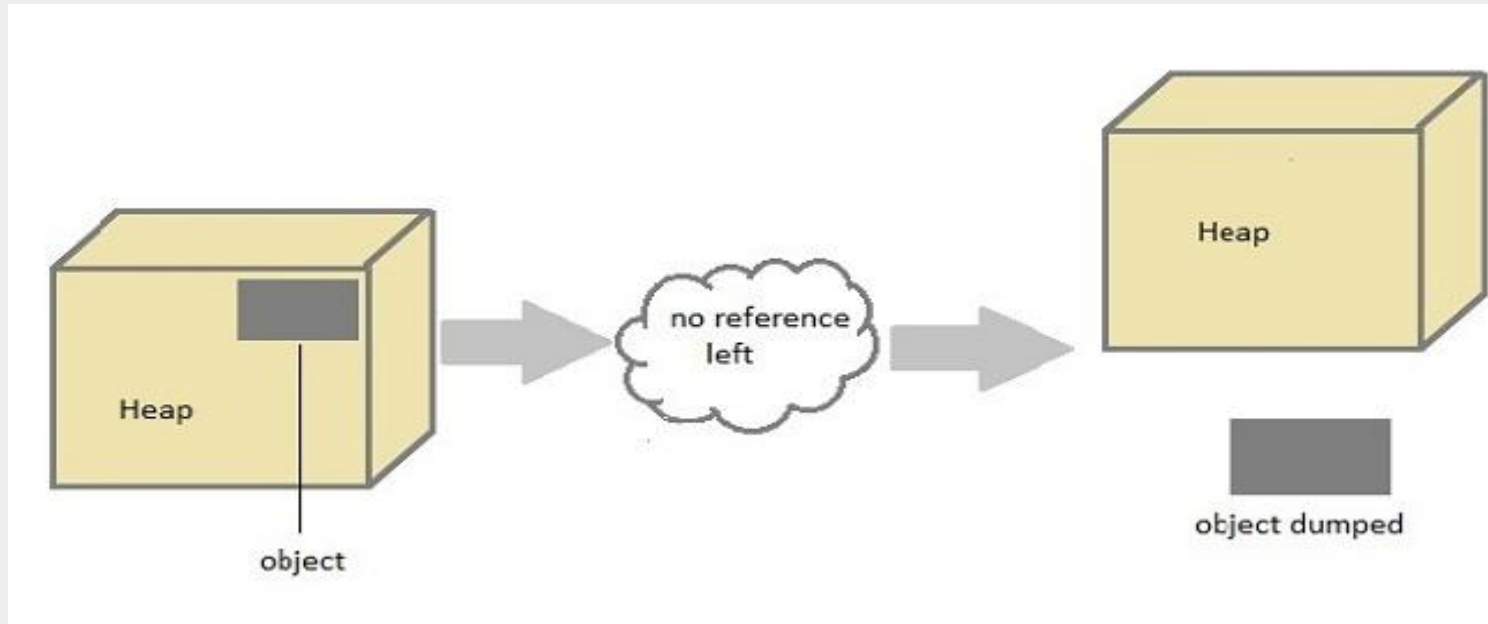
A decorative orange line graphic that starts as a horizontal line from the left edge, crosses the title, and then curves upwards and to the right, forming a large, open loop that occupies the upper right portion of the slide.

Memory Organization



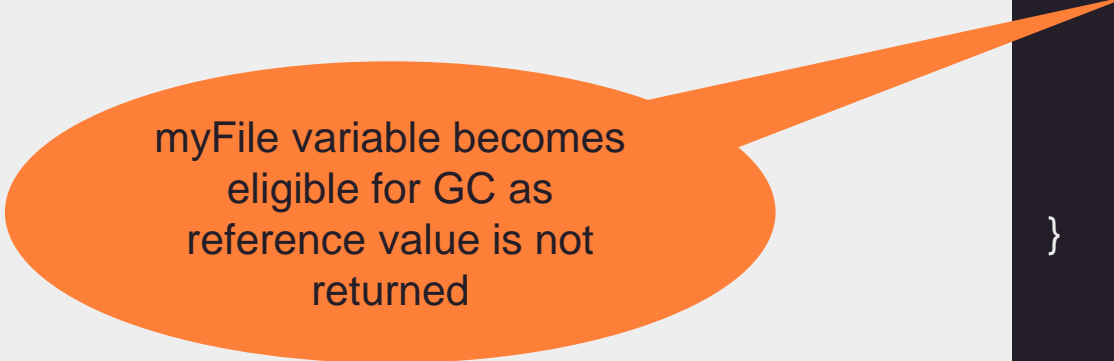
What is garbage collection ?

- In java, garbage refers to *unreferenced* objects.
- *Garbage Collection* is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
- The advantage of using Garbage Collection is ***better memory management***.



Facilitating Garbage Collection

- When method terminates
 - objects that are created and accessed by local references in a method are eligible for GC unless reference values to these objects are exported out of the method.



myFile variable becomes eligible for GC as reference value is not returned

```
import java.io.File;

public class GCTest {

    public void myMethod(){

        File myFile = new File("file.txt");

        //some code that accesses this myFile
        variable

    }

}
```

Facilitating Garbage Collection

- Other eligibility criteria for GC
 - when a method returned reference value is not assigned then the object is eligible for GC
 - if a reference is assigned a new reference value, the object prior to the assignment can become eligible for GC.
 - removing reachable references to a composite object can make the constituent objects become eligible for GC.

Demo : Box Class

- Consider a class named as Box as follows.

```
public class Box {  
    private float height;  
    private float width;  
    public Box() {  
        height=10;  
        width=10;  
    }  
    public Box(float height, float width) {  
        this.height = height;  
        this.width = width;  
    }  
}
```

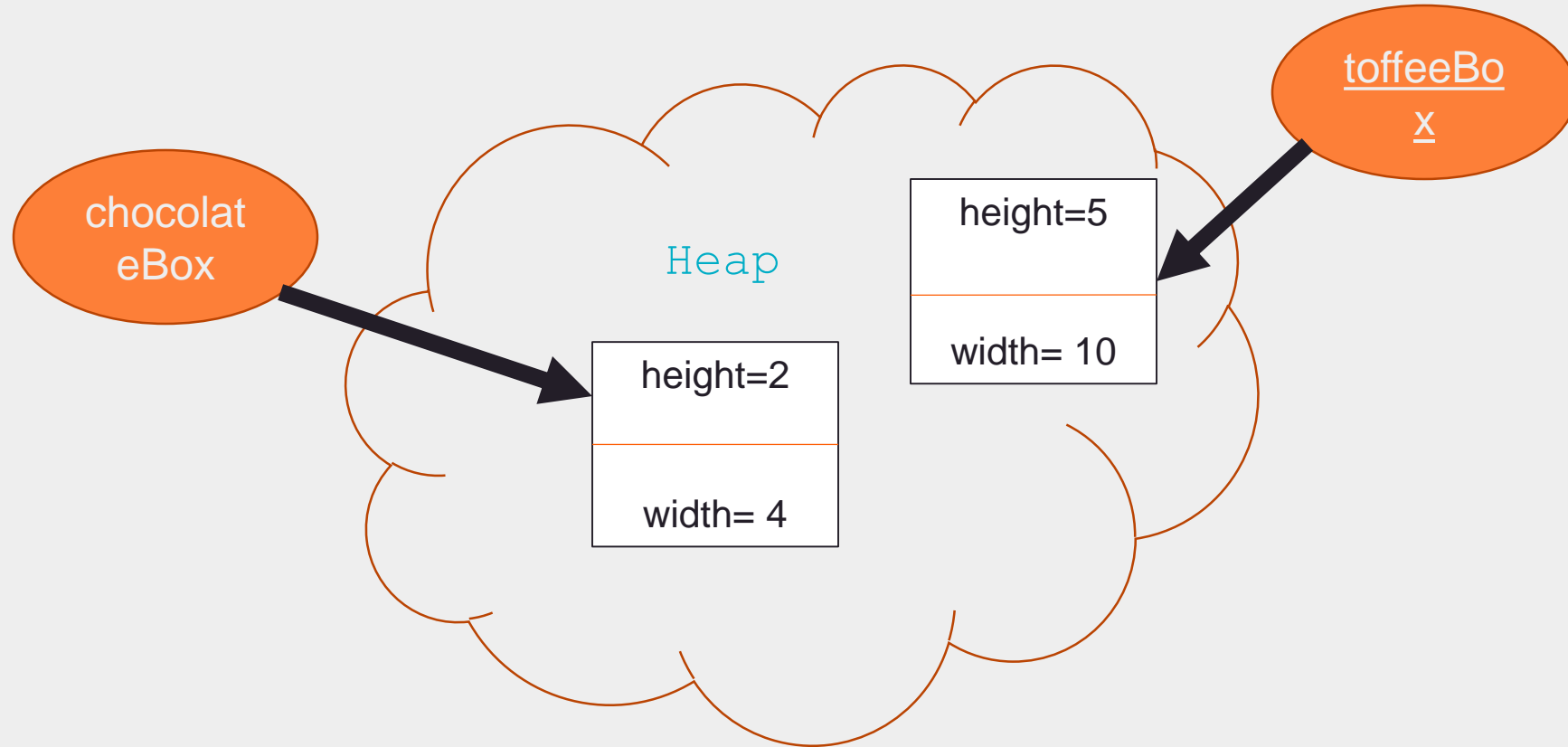
Demo : Phases of object lifetime

- Sample code for reference

```
public class BoxApplication {  
    public static void main(String[] args) {  
        //Memory Allocation  
        Box chocolateBox =new Box(2,4);  
        Box toffeeBox = new Box(5,10);  
  
        //Declaration  
        Box myBox;  
        //Memory Allocation  
        myBox=chocolateBox;  
  
        //Memory Allocation  
        chocolateBox=null;  
        myBox =null;  
  
    }  
}  
//Garbage Collection
```

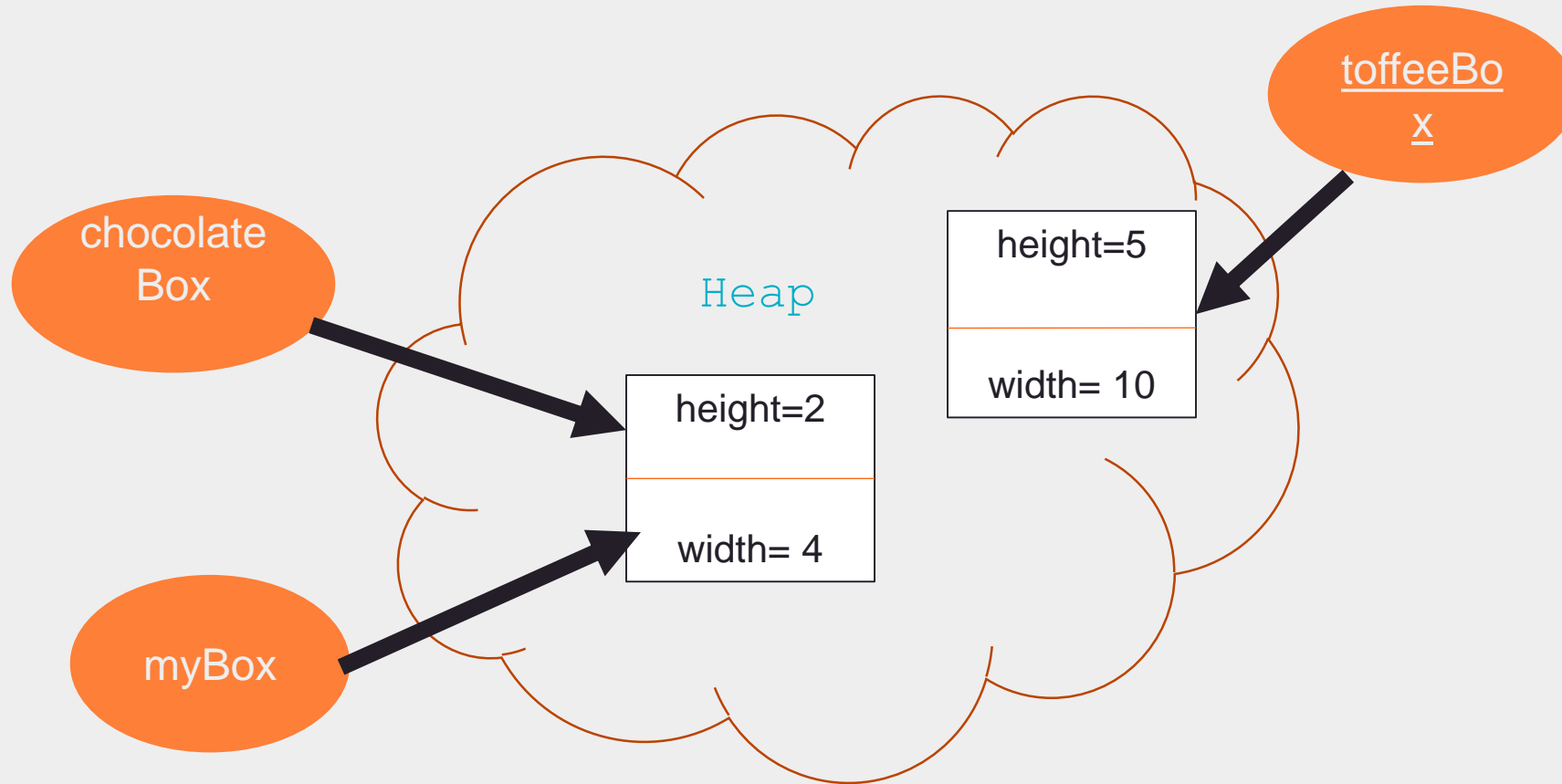

Memory Allocation

- Example



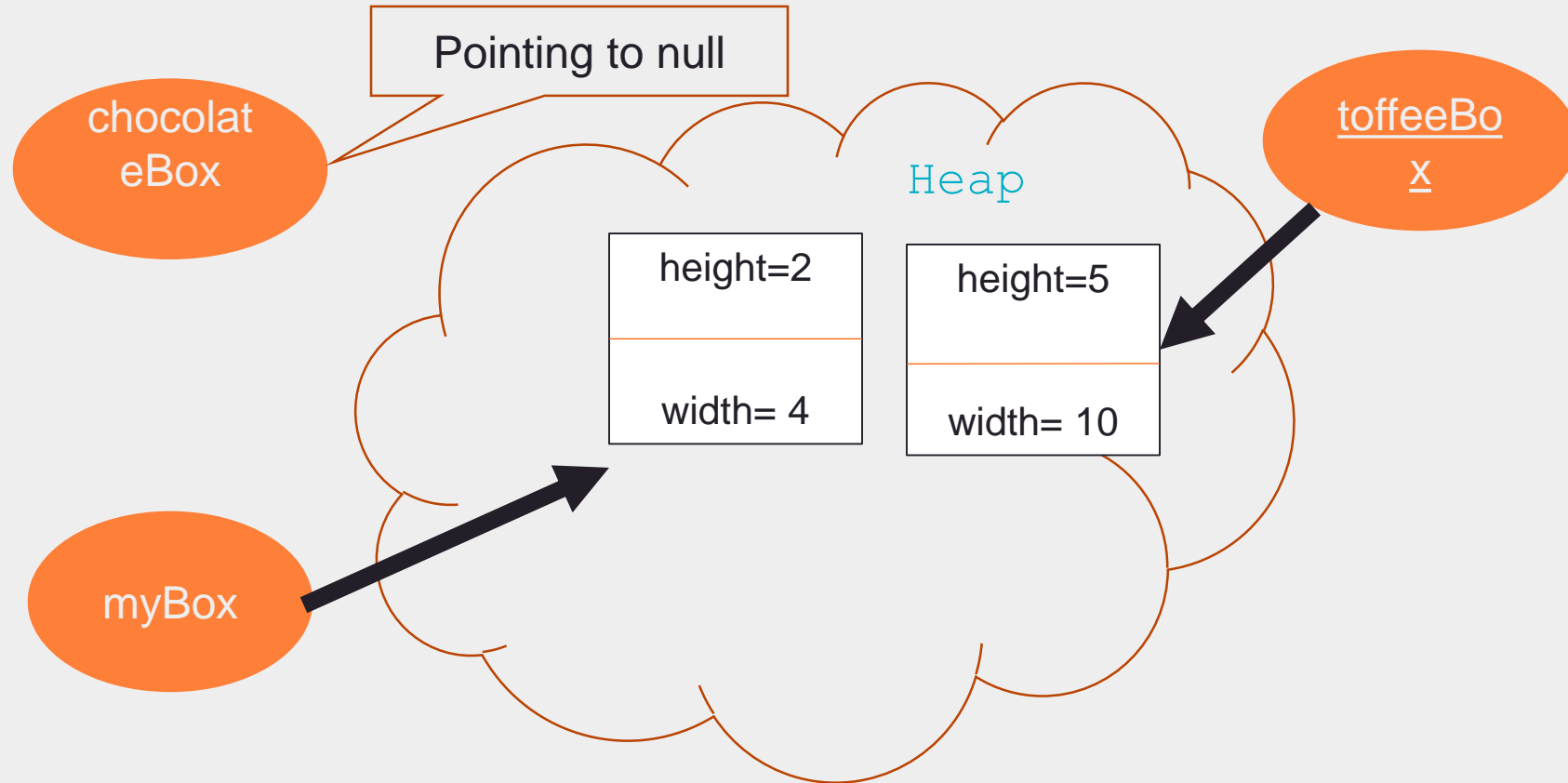
Memory Allocation

- Example



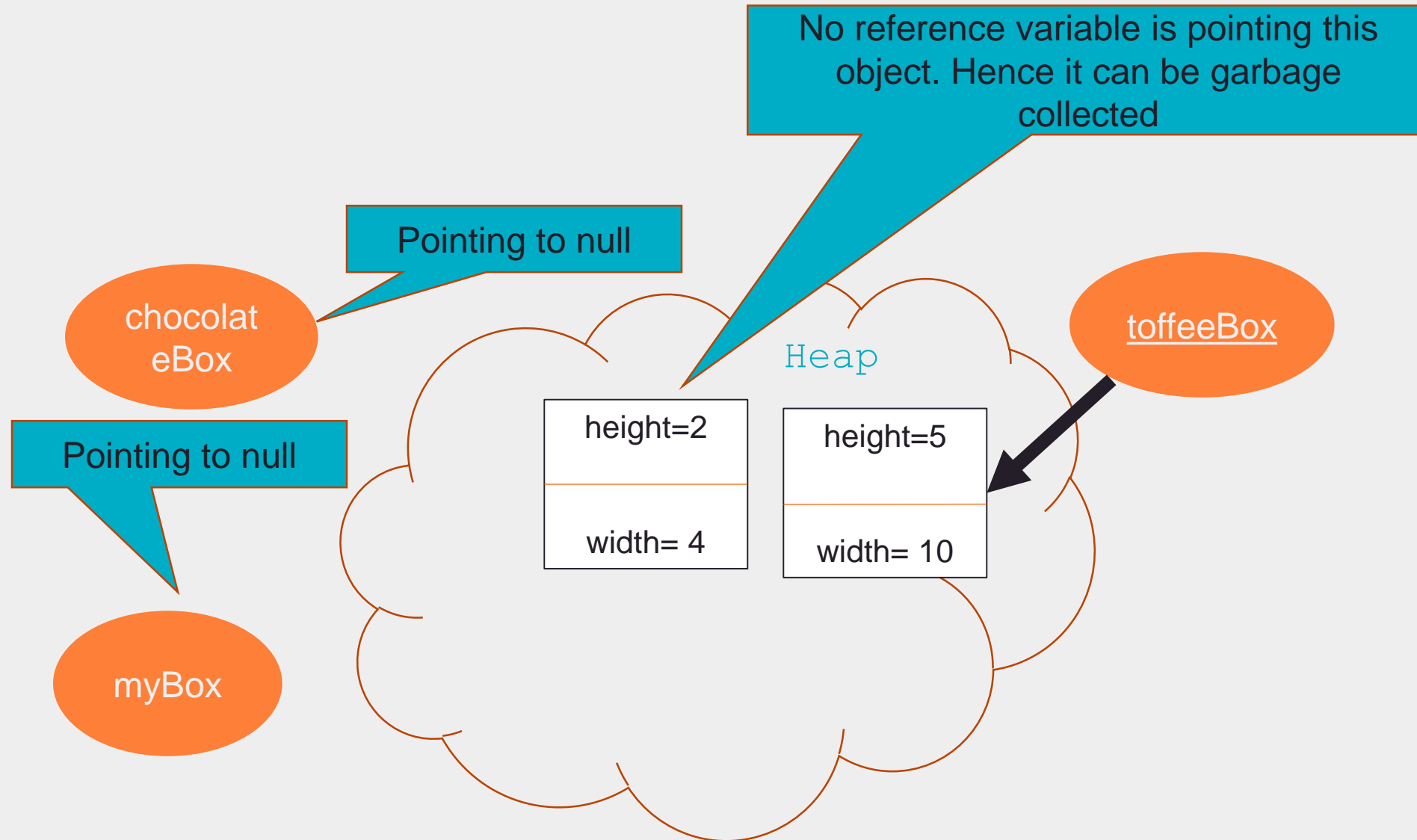
Memory Allocation

- Example



Memory Allocation

- Example



Garbage Collection

A decorative graphic consisting of a horizontal orange line that extends from the left edge of the slide to the right. At the point where the line reaches the right edge, it curves upwards and then forms a large, open circle that extends to the top and right edges of the slide.

How Garbage Collector works?

- Techniques based on
 - REFERENCE COUNTING
 - STOP-and-COPY
 - MARK-and-SWEEP

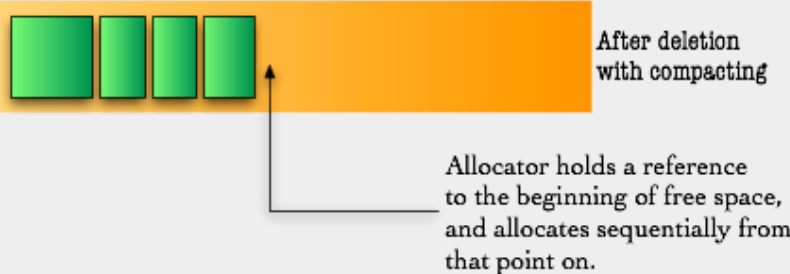
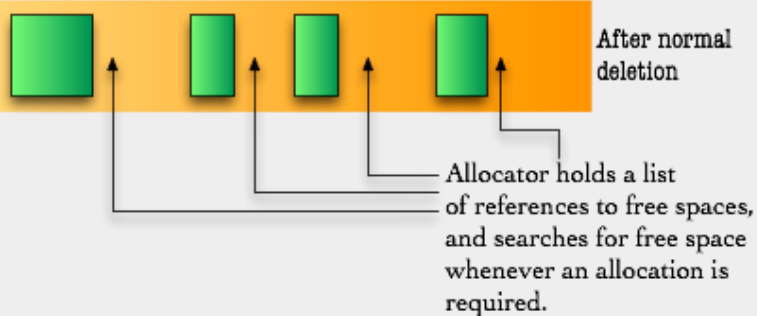
Algorithm: Marking stage

- In Mark and Sweep algorithm.
 - reachable objects are searched.
 - Any object which does not have reference are unreachable. They are eligible objects for GC.
 - As they are not getting used anymore, they are marked for garbage collection.

Garbage Collection : Sweeping

- The sweeping stage is where the deletion of objects take place.
 - Then the objects are moved closer to each other.
 - It removes any fragments of free space
 - This allows allocation much faster.

Garbage Collection



- A live object
- Marked for deletion
- Memory space

How to prompt GC to run?

- Garbage Collection cannot be forced to run. It will run only if more space is needed or if the collector wants to avoid running out of memory.
- To ask Garbage Collection to run use
 - `System.gc()` or
 - `Runtime.getRuntime().gc()`.

Invoking Garbage Collection

- The `System.gc()` method can be used to request garbage collection. However there is no guarantee that it will be executed.
- Java application has a unique `Runtime` object that can be used by the application to interact with the JVM. An application can obtain this object by calling the method `Runtime.getRuntime()`.

Invoking Garbage Collection

- Consider following example that shows how to invoke garbage collection.

```
public class GCTest {  
    final int SIZE = 50000;  
    void eatMemory() {  
        int[] intArray = new int[SIZE];  
        for (int i=0; i<SIZE; i++) {  
            intArray[i] = i;  
        }  
    }  
}
```

Invoking Garbage Collection

//Out put

free memory before creating array: 4988440

free memory after creating array: 4788424

free memory after running gc(): 5067600

```
public static void main (String[] args) {  
    GCTest gct = new GCTest();  
    // Step 1: get a Runtime object  
    Runtime r = Runtime.getRuntime();  
    // Step 2: determine the current amount of free memory  
    long freeMem = r.freeMemory();  
    System.out.println("free memory before creating array: " + freeMem);  
    // Step 3: consume some memory  
    gct.eatMemory();  
    // Step 4: determine amount of memory left after consumption  
    freeMem = r.freeMemory();  
    System.out.println("free memory after creating array: " + freeMem);  
    // Step 5: run the garbage collector, then check freeMemory  
    r.gc();  
    freeMem = r.freeMemory();  
    System.out.println("free memory after running gc(): " + freeMem);  
}}
```

Object Finalization

- `finalize()` method
 - is invoked before an object is removed by garbage collection.
 - to give it a last opportunity to clean up its act and free other kinds of resources it may be holding.
 - is guaranteed to be called once and only once.
 - can be overridden.

`protected void finalize() throws Throwable`

Finalizers to avoid

- Don't rely on finalizers to release non-memory resources.
- Finalizers may be run in any order.
- **So what are finalizers good for?**
 - to free memory allocated by native methods.
 - to provide a fallback mechanism for releasing non-memory finite resources such as file handles or sockets.

Exercise

Which of the following statements about finalize methods is/are true?

- a) The purpose of a finalize method is to recover system resources other than memory.
- b) The purpose of a finalize method is to recover memory.
- c) You should always write a finalize method for every class.
- d) The order in which objects are created controls the order in which their finalize methods are called.

Exercise

After which line the object initially referred by str is eligible for garbage collection?

```
1 class Garbage
2 {
3     public static void main(String arg[])
4     {
5         String str=new String("Hello");
6         String str1=str;
7         str=new String("Hi");
8         str1=new String("Hello Again");
9         return;
10    }
11 }
```

- a. After line no. 6
- b. After line no. 7
- c. After line no. 8
- d. After line no. 9

Summary

- With this we have come to an end of our session, where we discussed about
 - Object lifetime
 - Memory allocation in Java
 - Garbage collection in Java
- At the end of this session, we see that you are now able to answer following questions:
 - What is life cycle of an object
 - How Garbage Collection works in Java

Appendix

A decorative graphic consisting of a horizontal orange line that extends from the left edge of the slide. At its right end, it meets a vertical orange line that extends downwards to the bottom edge. A large, thin orange circle is positioned in the upper right quadrant, with its left edge touching the horizontal line and its bottom edge touching the vertical line.

References

Thank you

Reference Material : Websites & Blogs

- <http://www.javaworld.com/article/2076614/core-java/object-initialization-in-java.html>
- <https://docs.oracle.com/javase/tutorial/java/javaOO/objectcreation.html>
- http://docstore.mik.ua/orelly/java-ent/jnut/ch03_02.htm
- <http://www.javaworld.com/article/2077452/learn-java/java-memory-management.html>

Reference Material : Books

- ***Head First Java***
 - *By: Kathy Sierra, Bert Bates*
 - *Publisher: O'Reilly Media, Inc.*
- ***Java Complete Reference***
 - *By Herbert Schildt*



Persistent

Thank you!

Persistent University

